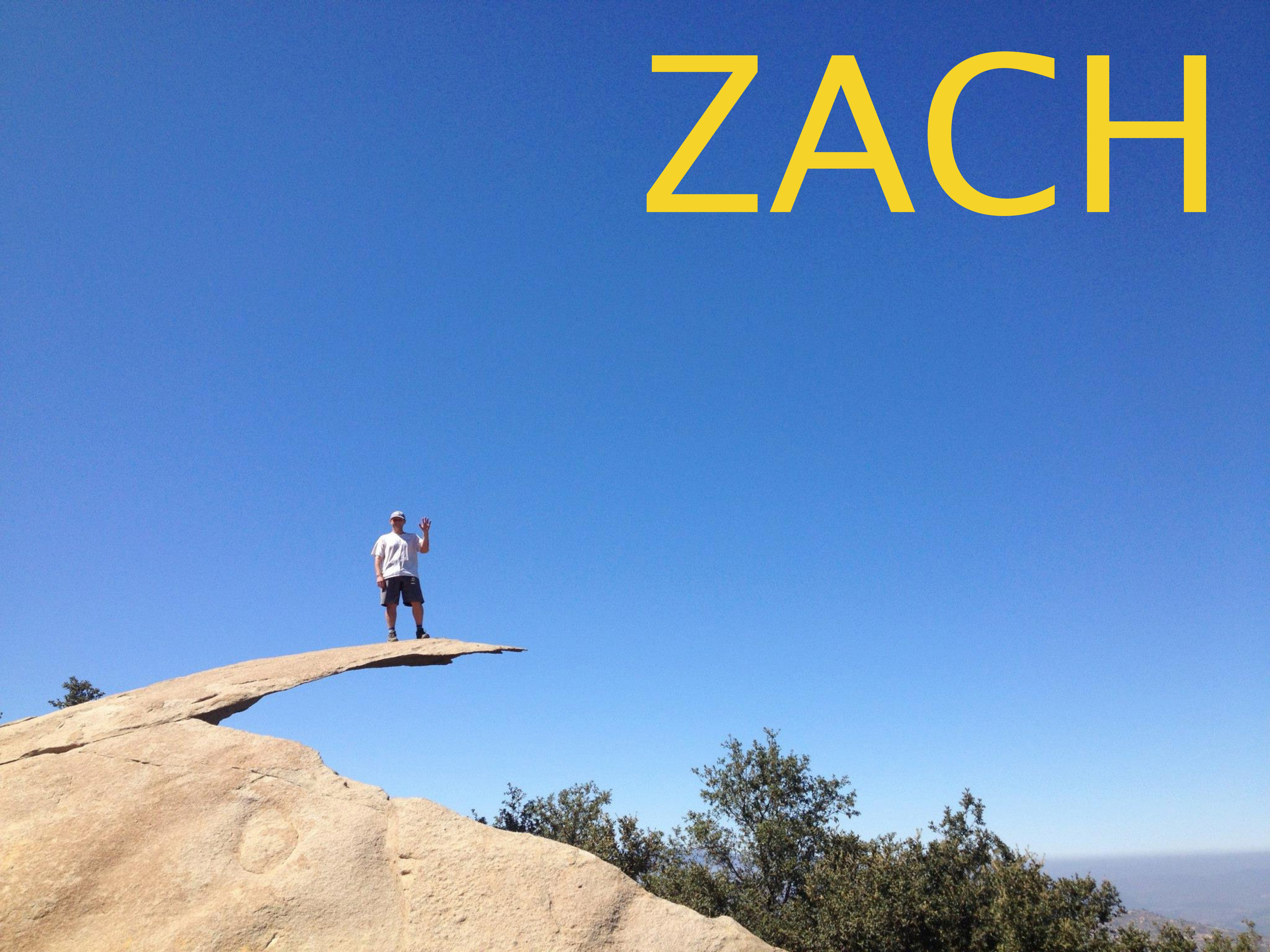
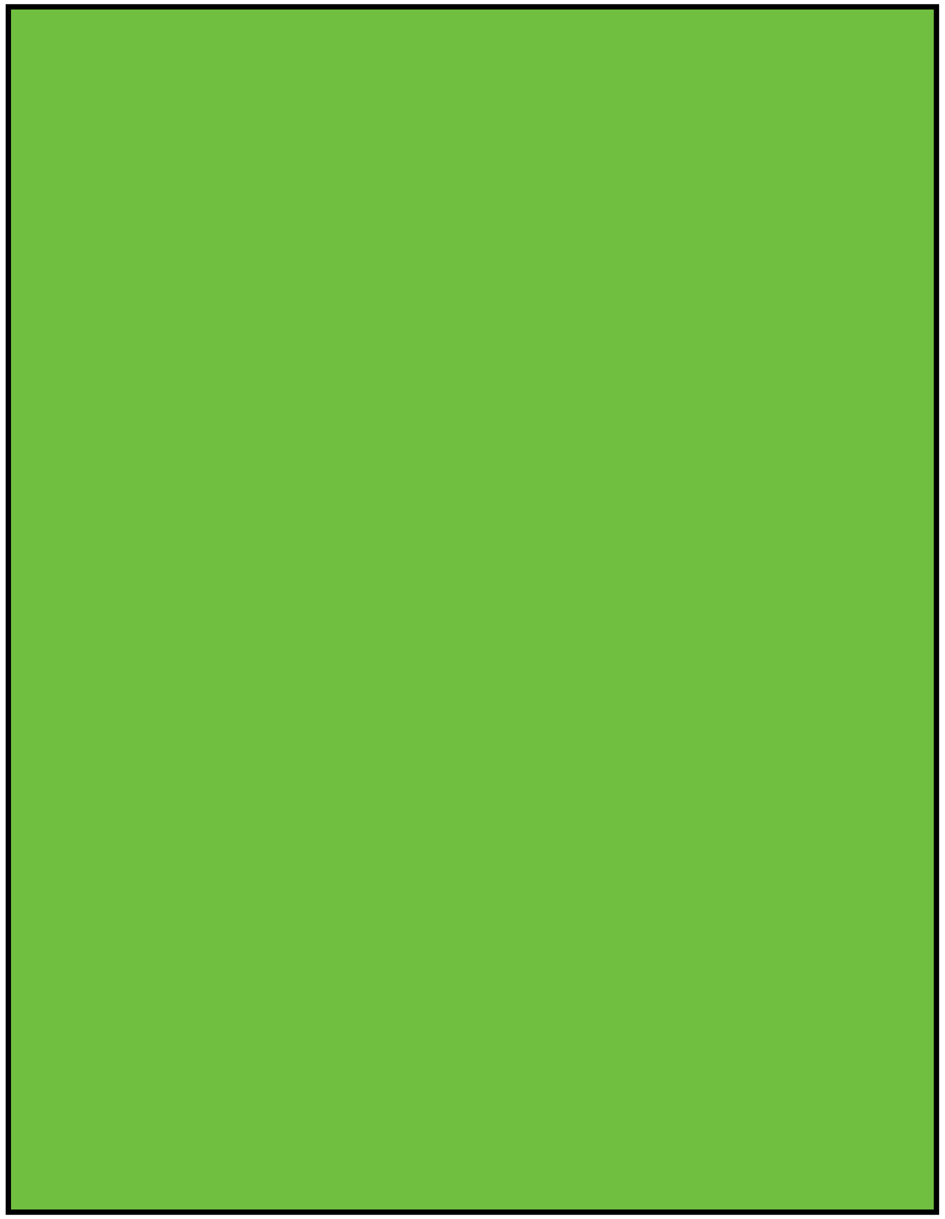
Normal Forms
+
DPLL

CSE 507
September 30, 2014


ZACH


DANGER
UNEXPLAINED NOTATION!
ASK QUESTIONS!
notación inexplicable
hacer preguntas
LOS VIOLADORES SERAN PROCESADOS



$$A ::= \top \mid \bot \mid x$$

Atoms

$$A ::= \top \mid \bot \mid x$$

$$A ::= \top \mid \bot \mid x$$

$$L ::= A \mid {\sim} A$$

```
A ::= ⊤ | ⊥ | x

L ::= A | ~ A
```

Literals

```
A ::= ⊤ | ⊥ | x

L ::= A | ~ A

F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
    |  F -> F
    |  F <-> F
```

```
A ::= ⊤ | ⊥ | x

L ::= A | ~ A

F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
    |  F -> F
    |  F <-> F
```

Formulas

```
A ::= ⊤ | ⊥ | x

L ::= A | ~ A

F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
    |  F -> F
    |  F <-> F
```

```
A ::= ⊤ | ⊥ | x

L ::= A | ~ A

F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
    |  F -> F
    |  F <-> F
```

A formula is satisfiable if there exists a function I from its variables to truth values such that, when we replace each variable x with I(x), the formula evaluates to true.

```
A ::= ⊤ | ⊥ | x

L ::= A | ~ A

F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
    |  F -> F
    |  F <-> F
```

A formula is satisfiable if there exists a function I from its variables to truth values such that, when we replace each variable x with I(x), the formula evaluates to true.

A formula is valid if for every function I from its variables to truth values, when we replace each variable x with I(x), the formula evaluates to true.

```
A ::= ⊤ | ⊥ | x

L ::= A | ~ A

F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
    |  F -> F
    |  F <-> F
```

A formula is satisfiable if there exists a function I from its variables to truth values such that, when we replace each variable x with I(x), the formula evaluates to true.

A formula is valid if for every function I from its variables to truth values, when we replace each variable x with I(x), the formula evaluates to true.

Formula f is valid **iff** **!f** **is**

```
A ::= ⊤ | ⊥ | x

L ::= A | ~ A

F ::= L
    | ! F
    | F /\ F
    | F \/ F
    | F -> F
    | F <-> F
```

A formula is satisfiable if there exists a function I from its variables to truth values such that, when we replace each variable x with I(x), the formula evaluates to true.

A formula is valid if for every function I from its variables to truth values, when we replace each variable x with I(x), the formula evaluates to true.

Formula f is valid **iff !f is**

We can try to determine validity by search (enumerating assignments I) or by

Sat Solver

Sat Solver

```
sat f =
   ...
```

Sat Solver

define function `sat` that takes formula `f` as an

```
sat f =
    ...
```

Sat Solver

```
sat f =
   case f of
   | ...
```

Sat Solver

```
sat f =
   case f of
   | ...
```

analyze structure of `f`

Sat Solver

```
sat f =
   case f of
   | ⊤  => SAT
   | ...
```

Sat Solver

```
sat f =
   case f of
   | ⊤  => SAT
   | ...
```

if `f` is just true,

Sat Solver

```
sat f =
   case f of
   | ⊤  => SAT
   | ⊥  => UNSAT
   | ...
```

Sat Solver

```
sat f =
   case f of
    | ⊤  => SAT
    | ⊥  => UNSAT
    | ...
```

if f is just false,

Sat Solver

```
sat f =
   case f of
   | ⊤  => SAT
   | ⊥  => UNSAT
   | _  =>
      ...
```

Sat Solver

```
sat f =
    case f of
    | ⊤  => SAT
    | ⊥  => U
    | _  =>
       ...
```

all other cases...

Sat Solver

```
sat f =
   case f of
   | ⊤  => SAT
   | ⊥  => UNSAT
   | _  =>
     x = pick_var f
     ...
```

Sat Solver

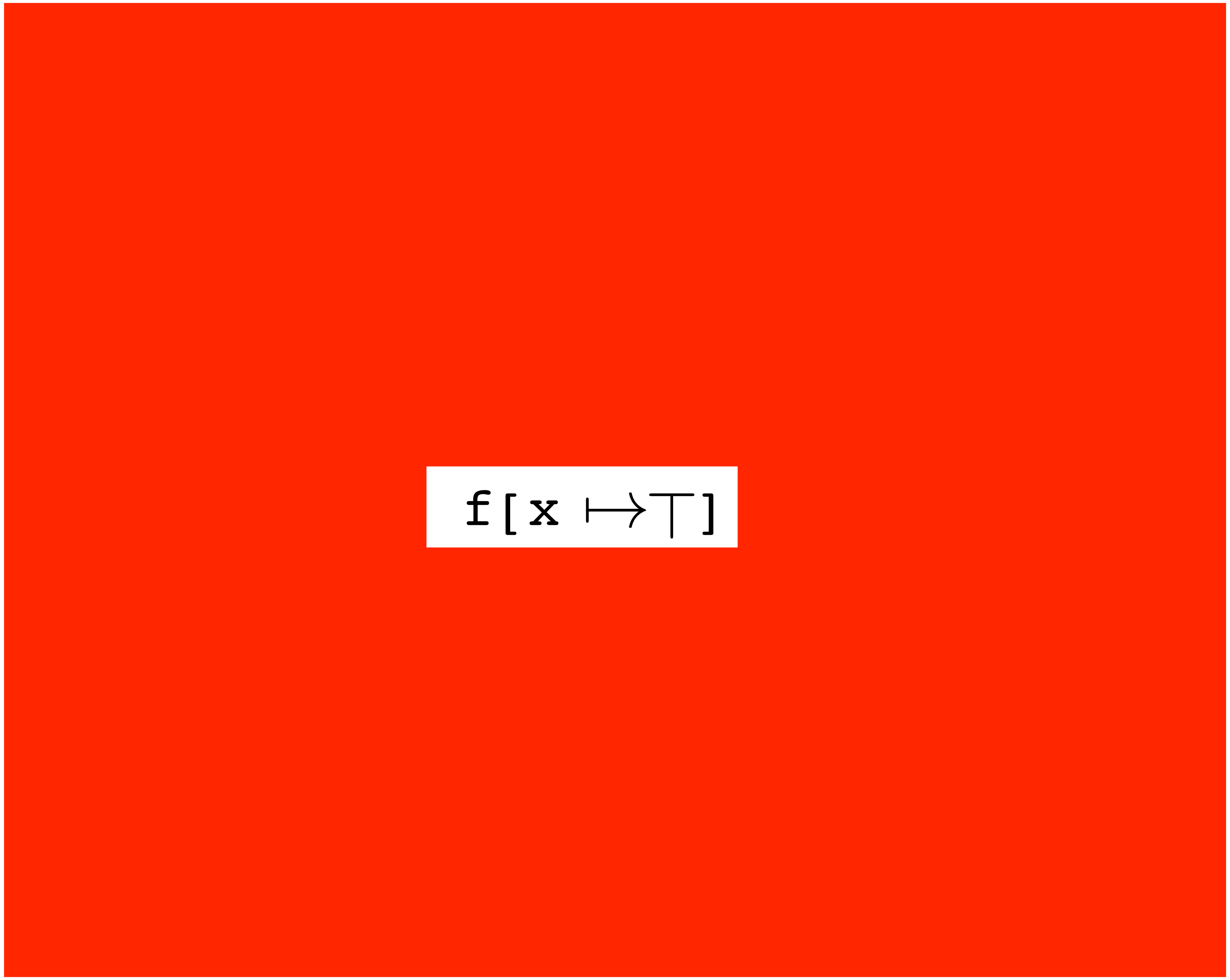
```
sat f =
   case f of
    | ⊤  => SAT
    | ⊥  => UNSAT
    | _  =>
      x = pick_var f
      ...
```

choose some variable from `f`

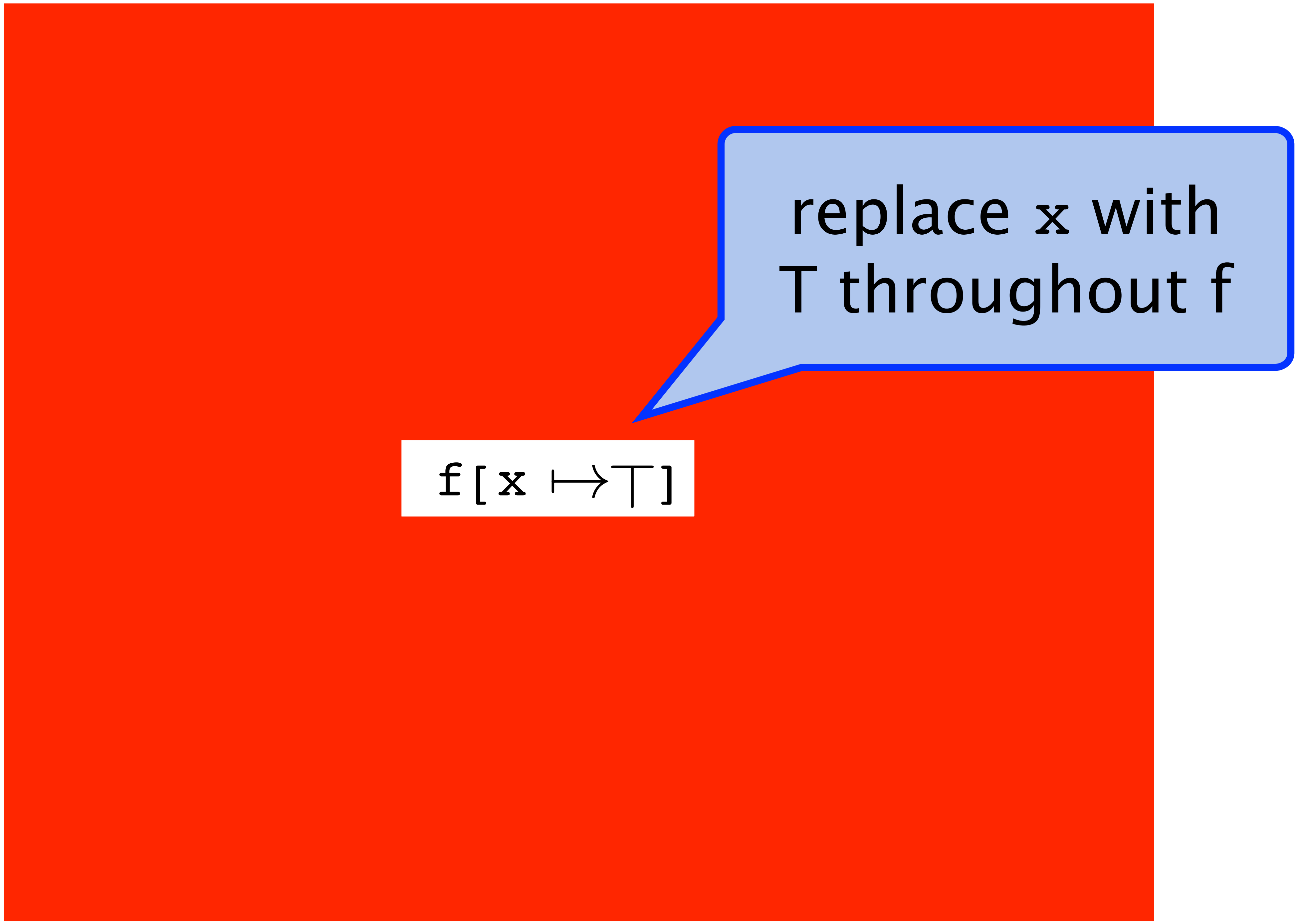
Sat Solver

```
sat f =
   case f of
   | ⊤  => SAT
   | ⊥  => UNSAT
   | _  =>
      x = pick_var f
      if sat f[x ↦ ⊤] = SAT then
         ...
```

Sat Solver



Sat Solver



Sat Solver

$f[x \mapsto \top]$

replace x with T throughout f

(x /\ y \/ z -> y \/ x)[x↦ T] = (T /\ y \/ z -> y \/ T)

Sat Solver

```
sat f =
   case f of
   | ⊤  => SAT
   | ⊥  => UNSAT
   | _  =>
     x = pick_var f
     if sat f[x ↦ ⊤] = SAT then
        ...
```

Sat Solver

```
sat f =
   case f of
    | ⊤  => SAT
    | ⊥  => UNSAT
    | _  =>
       x = pick_var f
       if sat f[x ↦ ⊤] = SAT then
          ...
```

if we find a satisfying assignment with x set to T

Sat Solver

```
sat f =
   case f of
   | ⊤  => SAT
   | ⊥  => UNSAT
   | _  =>
      x = pick_var f
      if sat f[x ↦ ⊤] = SAT then
         SAT
      else
         ...
```

Sat Solver

```
sat f =
   case f of
   | ⊤  => SAT
   | ⊥  => UNSAT
   | _  =>
      x = pick_var f
      if sat f[x ↦ ⊤] = SAT then
         SAT
      else
         ...
```

then formula f is

Sat Solver

```
sat f =
   case f of
   | ⊤  => SAT
   | ⊥  => UNSAT
   | _  =>
      x = pick_var f
      if sat f[x ↦ ⊤] = SAT then
         SAT
      else
         sat (f[x ↦ ⊥])
```

Sat Solver

```
sat f =
   case f of
   | ⊤  => SAT
   | ⊥  => UNSAT
   | _  =>
     x = pick_var f
     if sat
        SAT
     else
        sat (f[x ↦ ⊥])
```

try setting x to false ...

... and recurse

Sat Solver

```
sat f =
   case f of
   | ⊤  => SAT
   | ⊥  => UNSAT
   | _  =>
      x = pick_var f
      if sat f[x ↦ ⊤] = SAT then
         SAT
      else
         sat (f[x ↦ ⊥])
```

Correct?

Sat Solver

```
sat f =
   case f of
   | ⊤  => SAT
   | ⊥  => UNSAT
   | _  =>
      x = pick_var f
      if sat f[x ↦ ⊤] = SAT then
         SAT
      else
         sat (f[x ↦ ⊥])
```

Sat Solver

```
sat f =
   case f of
   | ⊤  => SAT
   | ⊥  => UNSAT
   | _  =>
      x = pick_var f
      if sat f[x ↦ ⊤] = SAT then
         SAT
      else
         sat (f[x ↦ ⊥])
```

Today

Normal Forms
- desugaring
- negation normal form (NNF)
- disjunctive normal form (DNF)
- conjuctive normal form (CNF)

Today

Normal Forms
- desugaring
- negation normal form (NNF)
- disjunctive normal form (DNF)
- conjuctive normal form (CNF)

DPLL
- resolution
- binary constraint propagation
- a better sat solver

Today

clean up

Normal Forms
- desugaring
- negation normal form (NNF)
- disjunctive normal form (DNF)
- conjuctive normal form (CNF)

DPLL
- resolution
- binary constraint propagation
- a better sat solver

Today

clean up

Normal Forms
- desugaring
- negation normal form (NNF)
- disjunctive normal form (DNF)
- conjuctive normal form (CNF)

DPLL

combine search and

- resolution
- binary constraint propagation
- a better sat solver

```
A ::= ⊤ | ⊥ | x

L ::= A | ~ A

F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
    |  F -> F
    |  F <-> F
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
    |  F -> F
    |  F <-> F
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
    |  F -> F
    |  F <-> F
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
    |  F -> F
    |  F <-> F
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
    |  F -> F
    |  F <-> F
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
```

```
desugar f =
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
    |  F -> F
    |  F <-> F
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
```

```
desugar f =
  case f of
  | l => l
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
    |  F -> F
    |  F <-> F
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
```

```
desugar f =
  case f of
  | l => l
  | f1 /\ f2 => (desugar f1) /\ (desugar f2)
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
    |  F -> F
    |  F <-> F
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
```

```
desugar f =
  case f of
  | l => l
  | f1 /\ f2 => (desugar f1) /\ (desugar f2)
  | f1 \/ f2 => (desugar f1) \/ (desugar f2)
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
    |  F -> F
    |  F <-> F
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
```

```
desugar f =
   case f of
   | l => l
   | f1 /\ f2 => (desugar f1) /\ (desugar f2)
   | f1 \/ f2 => (desugar f1) \/ (desugar f2)
   | f1 -> f2 => desugar ((! f1) \/ f2)
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
    |  F -> F
    |  F <-> F
```

→

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
```

```
desugar f =
  case f of
  | l => l
  | f1 /\ f2 => (desugar f1) /\ (desugar f2)
  | f1 \/ f2 => (desugar f1) \/ (desugar f2)
  | f1 -> f2 => desugar ((! f1) \/ f2)
  | f1 <-> f2 => desugar ((f1 -> f2) /\ (f2 -> f1))
```

```
F ::= L
    |   ! F
    |   F /\ F
    |   F \/ F
    |   F -> F
    |   F <-> F
```

➡

```
F ::= L
    |   ! F
    |   F /\ F
    |   F \/ F
```

Correct? Efficient?

```
desugar f =
   case f of
   | l => l
   | f1 /\ f2 => (desugar f1) /\ (desugar f2)
   | f1 \/ f2 => (desugar f1) \/ (desugar f2)
   | f1 -> f2 => desugar ((! f1) \/ f2)
   | f1 <-> f2 => desugar ((f1 -> f2) /\ (f2 -> f1))
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
```

```
F ::= L
    |  F /\ F
    |  F \/ F
```

NNF

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
```

```
F ::= L
    |  F /\ F
    |  F \/ F
```

NNF

```
pnot f =
```

```
F ::= L
   |  ! F
   |  F /\ F
   |  F \/ F
```

```
F ::= L
   |  F /\ F
   |  F \/ F
```

NNF

```
pnot f =
  case f of
  | ! (~ a) => a
```

```
F ::= L
   |  ! F
   |  F /\ F
   |  F \/ F
```

```
F ::= L
   |  F /\ F
   |  F \/ F
```

NNF

```
pnot f =
   case f of
   | ! (~ a) => a
   | ! a     => ~ a
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
```

```
F ::= L
    |  F /\ F
    |  F \/ F
```

NNF

```
pnot f =
   case f of
   | ! (~ a) => a
   | ! a     => ~ a
   | ! ! f   => pnot f
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
```

```
F ::= L
    |  F /\ F
    |  F \/ F
```

NNF

```
pnot f =
   case f of
   | ! (~ a) => a
   | ! a     => ~ a
   | ! ! f   => pnot f
   | ! (f1 /\ f2) => pnot ((! f1) \/ (! f2))
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
```

→

```
F ::= L
    |  F /\ F
    |  F \/ F
```

NNF

```
pnot f =
   case f of
   | ! (~ a) => a
   | ! a     => ~ a
   | ! ! f   => pnot f
   | ! (f1 /\ f2) => pnot ((! f1) \/ (! f2))
   | ! (f1 \/ f2) => pnot ((! f1) /\ (! f2))
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
```

```
F ::= L
    |  F /\ F
    |  F \/ F
```

NNF

```
pnot f =
   case f of
   | ! (~ a)  => a
   | ! a      => ~ a
   | ! ! f    => pnot f
   | ! (f1 /\ f2) => pnot ((! f1) \/ (! f2))
   | ! (f1 \/ f2) => pnot ((! f1) /\ (! f2))
   | f1 /\ f2     => (pnot f1) /\ (pnot f2)
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
```

```
F ::= L
    |  F /\ F
    |  F \/ F
```

NNF

```
pnot f =
   case f of
   | ! (~ a) => a
   | ! a     => ~ a
   | ! ! f   => pnot f
   | ! (f1 /\ f2) => pnot ((! f1) \/ (! f2))
   | ! (f1 \/ f2) => pnot ((! f1) /\ (! f2))
   | f1 /\ f2     => (pnot f1) /\ (pnot f2)
   | f1 \/ f2     => (pnot f1) \/ (pnot f2)
```

```
F ::= L
    |  ! F
    |  F /\ F
    |  F \/ F
```

→

```
F ::= L
    |  F /\ F
    |  F \/ F
```

NNF

```
pnot f =
   case f of
   | ! (~ a)    => a
   | ! a        => ~ a
   | ! ! f      => pnot f
   | ! (f1 /\ f2) => pnot ((! f1) \/ (! f2))
   | ! (f1 \/ f2) => pnot ((! f1) /\ (! f2))
   | f1 /\ f2     => (pnot f1) /\ (pnot f2)
   | f1 \/ f2     => (pnot f1) \/ (pnot f2)
```

Correct? Efficient?

```
F ::= L
    |  F /\ F
    |  F \/ F
```

```
F ::= L
   |  F /\ F
   |  F \/ F
```

```
F ::= L
    | F /\ F
    | F \/ F
```

→

```
F ::= C | C \/ F
C ::= L | L /\ C
```

DNF

```
F ::= L
    | F /\ F
    | F \/ F
```

```
F ::= C | C \/ F
C ::= L | L /\ C
```

DNF

```
dnf f =
```

```
F ::= L
    | F /\ F
    | F \/ F
```

```
F ::= C | C \/ F
C ::= L | L /\ C
```

DNF

```
dnf f =
   case f of
   | l => l
```

```
F ::= L
    |  F /\ F
    |  F \/ F
```

```
F ::= C | C \/ F
C ::= L | L /\ C
```

DNF

```
dnf f =
   case f of
   | l => l
   | (f1 \/ f2) /\ f3 => dnf ((f1 /\ f3) \/ (f2 /\ f3))
```

```
F ::= L
    | F /\ F
    | F \/ F
```

→

```
F ::= C | C \/ F
C ::= L | L /\ C
```

DNF

```
dnf f =
   case f of
   | l => l
   | (f1 \/ f2) /\ f3 => dnf ((f1 /\ f3) \/ (f2 /\ f3))
   | f1 /\ (f2 \/ f3) => dnf ((f1 /\ f2) \/ (f1 /\ f3))
```

```
F ::= L
    |  F /\ F
    |  F \/ F
```

```
F ::= C | C \/ F
C ::= L | L /\ C
```

DNF

```
dnf f =
   case f of
   | l => l
   | (f1 \/ f2) /\ f3 => dnf ((f1 /\ f3) \/ (f2 /\ f3))
   | f1 /\ (f2 \/ f3) => dnf ((f1 /\ f2) \/ (f1 /\ f3))
   | f1 /\ f2 => (dnf f1) X (dnf f2)
```

```
F ::= L
    | F /\ F
```

```
F ::= C | C \/ F
C ::= L | L /\ C
```

$(a_1 \lor \ldots \lor a_k) \times (b_1 \lor \ldots \lor b_n) =$

$(a_1 \land b_1) \lor \ldots \lor$
$(a_i \land b_j) \lor \ldots \lor$
$(a_k \land b_n)$

```
dn
  | f
  | (f1 \/ f2)                  ) \/ (f2 /\ f3))
  | f1 /\ (f2 \/ f3) =>  ((f1 /\ f2) \/ (f1 /\ f3))
  | f1 /\ f2 => (dnf f1) X (dnf f2)
```

```
F ::= L
    |  F /\ F
    |  F \/ F
```

```
F ::= C | C \/ F
C ::= L | L /\ C
```

DNF

```
dnf f =
   case f of
    | l => l
    | (f1 \/ f2) /\ f3 => dnf ((f1 /\ f3) \/ (f2 /\ f3))
    | f1 /\ (f2 \/ f3) => dnf ((f1 /\ f2) \/ (f1 /\ f3))
    | f1 /\ f2 => (dnf f1) X (dnf f2)
    | f1 \/ f2 => (dnf f1) \/ (dnf f2)
```

```
F ::= L
    | F /\ F
    | F \/ F
```

→

```
F ::= C | C \/ F
C ::= L | L /\ C
```

DNF

Correct? Efficient?

```
dnf f =
   case f of
    | l => l
    | (f1 \/ f2) /\ f3 => dnf ((f1 /\ f3) \/ (f2 /\ f3))
    | f1 /\ (f2 \/ f3) => dnf ((f1 /\ f2) \/ (f1 /\ f3))
    | f1 /\ f2 => (dnf f1) X (dnf f2)
    | f1 \/ f2 => (dnf f1) \/ (dnf f2)
```

```
F ::= L
    |  F /\ F
    |  F \/ F
```

```
F ::= L
    |  F /\ F
    |  F \/ F
```

```
F ::= L
   |  F /\ F
   |  F \/ F
```

```
F ::= D | D /\ C
D ::= L | L \/ D
```

CNF

```
F ::= L
    | F /\ F
    | F \/ F
```

```
F ::= D | D /\ C
D ::= L | L \/ D
```

CNF

```
cnf f =
```

```
F ::= L
    |  F /\ F
    |  F \/ F
```

```
F ::= D | D /\ C
D ::= L | L \/ D
```

CNF

```
cnf f =
   case f of
   | l => l
```

```
F ::= L
    |  F /\ F
    |  F \/ F
```

```
F ::= D | D /\ C
D ::= L | L \/ D
```

CNF

```
cnf f =
   case f of
   | l => l
   | (f1 /\ f2) \/ f3 => cnf ((f1 \/ f3) /\ (f2 \/ f3))
```

```
F ::= L
    | F /\ F
    | F \/ F
```

→

```
F ::= D | D /\ C
D ::= L | L \/ D
```

CNF

```
cnf f =
   case f of
    | l => l
    | (f1 /\ f2) \/ f3 => cnf ((f1 \/ f3) /\ (f2 \/ f3))
    | f1 \/ (f2 /\ f3) => cnf ((f1 \/ f2) /\ (f1 \/ f3))
```

```
F ::= L
    | F /\ F
    | F \/ F
```

```
F ::= D | D /\ C
D ::= L | L \/ D
```

CNF

```
cnf f =
   case f of
   | l => l
   | (f1 /\ f2) \/ f3 => cnf ((f1 \/ f3) /\ (f2 \/ f3))
   | f1 \/ (f2 /\ f3) => cnf ((f1 \/ f2) /\ (f1 \/ f3))
   | f1 \/ f2 => (cnf f1) X (cnf f2)
```

```
F ::= L
    | F /\ F
```

```
F ::= D | D /\ C
D ::= L | L \/ D
```

```
(a_1 /\ ... /\ a_k) X (b_1 /\ ... /\ b_n) =

    (a_1 \/ b_1) /\ ... /\
    (a_i \/ b_j) /\ ... /\
    (a_k \/ b_n)
```

```
cn
  | f
  | (f1 /\ f2)                 f3) /\ (f2 \/ f3))
  | f1 \/ (f2 /\ f3) =>      (f1 \/ f2) /\ (f1 \/ f3))
  | f1 \/ f2 => (cnf f1) X (cnf f2)
```

```
F ::= L
    | F /\ F
    | F \/ F
```

```
F ::= D | D /\ C
D ::= L | L \/ D
```

CNF

```
cnf f =
   case f of
   | l => l
   | (f1 /\ f2) \/ f3 => cnf ((f1 \/ f3) /\ (f2 \/ f3))
   | f1 \/ (f2 /\ f3) => cnf ((f1 \/ f2) /\ (f1 \/ f3))
   | f1 \/ f2 => (cnf f1) X (cnf f2)
   | f1 /\ f2 => (cnf f1) /\ (cnf f2)
```

```
F ::= L
   |  F /\ F
   |  F \/ F
```

```
F ::= D | D /\ C
D ::= L | L \/ D
```

CNF

Correct? Efficient?

```
cnf f =
   case f of
    | l => l
    | (f1 /\ f2) \/ f3 => cnf ((f1 \/ f3) /\ (f2 \/ f3))
    | f1 \/ (f2 /\ f3) => cnf ((f1 \/ f2) /\ (f1 \/ f3))
    | f1 \/ f2 => (cnf f1) X (cnf f2)
    | f1 /\ f2 => (cnf f1) /\ (cnf f2)
```

```
F ::= L
    |  F /\ F
    |  F \/ F
```

```
F ::= D | D /\ C
D ::= L | L \/ D
```

CNF

Correct? Efficient?

```
cnf f =
   case f of
   | l => l
   | (f1 /\ f2) \/ f3 => cnf ((f1 \/ f3) /\ (f2 \/ f3))
   | f1 \/ (f2 /\ f3) => cnf ((f1 \/ f2) /\ (f1 \/ f3))
   | f1 \/ f2 => (cnf f1) X (cnf f2)
   | f1 /\ f2 => (cnf f1) /\ (cnf f2)
```

```
F ::= L
   |  F /\ F
   |  F \/ F
```

```
F ::= D | D /\ C
D ::= L | L \/ D
```

```
F ::= L
    |  F /\ F
    |  F \/ F
```

```
F ::= D | D /\ C
D ::= L | L \/ D
```

```
cnf f =
```

```
F ::= L
    |  F /\ F
    |  F \/ F
```

```
F ::= D | D /\ C
D ::= L | L \/ D
```

```
cnf f =
   case f of
   | l => l
```

```
F ::= L
    | F /\ F
    | F \/ F
```

```
F ::= D | D /\ C
D ::= L | L \/ D
```

```
cnf f =
   case f of
   | l => l
   | f1 /\ f2 => (cnf f1) /\ (cnf f2)
```

```
F ::= L
    | F /\ F
    | F \/ F
```

```
F ::= D | D /\ C
D ::= L | L \/ D
```

```
cnf f =
   case f of
   | l => l
   | f1 /\ f2 => (cnf f1) /\ (cnf f2)
   | f1 \/ f2 =>
```

```
F ::= L
    | F /\ F
    | F \/ F
```

→

```
F ::= D | D /\ C
D ::= L | L \/ D
```

```
cnf f =
   case f of
   | l => l
   | f1 /\ f2 => (cnf f1) /\ (cnf f2)
   | f1 \/ f2 =>
         (x1 \/ x2) /\
         (distr_or (~ x1) (cnf f1)) /\
         (distr_or (~ x2) (cnf f2))
```

```
F ::= L
```

```
F ::= D | D /\ C
```

```
(distr_or (~ x) ((d1,1 \/ ... \/ d1,k) /\
                 ... /\
                 (dm,1 \/ ... \/ dm,n)) =

  ((~x \/ d1,1 \/ ... \/ d1,k) /\
   ... /\
   (~x \/ dm,1 \/ ... \/ dm,n))
```

```
(cnf f2)
```

```
(distr_or (~ x1) (cnf f1)) /\
(distr_or (~ x2) (cnf f2))
```

```
F ::= L
   |  F /\ F
   |  F \/ F
```

→

```
F ::= D | D /\ C
D ::= L | L \/ D
```

```
cnf f =
   case f of
   | l => l
   | f1 /\ f2 => (cnf f1) /\ (cnf f2)
   | f1 \/ f2 =>
        (x1 \/ x2) /\
        (distr_or (~ x1) (cnf f1)) /\
        (distr_or (~ x2) (cnf f2))
```

Resolution

$$
\frac{\begin{array}{l} C_a : (a_1 \lor a_2 \lor \ldots \lor p \lor \ldots \lor a_k) \\ C_b : (b_1 \lor b_2 \lor \ldots \lor \sim p \lor \ldots \lor b_n) \end{array}}{(a_1 \lor \ldots \lor a_k \lor b_1 \lor \ldots \lor b_n)}
$$

Unit Resolution

C_a : p

C_b : (b_1 \/ b_2 \/ ... \/ ~p \/ ... \/ b_n)

(b_1 \/ ... \/ b_n)

Unit Resolution

C_a : p

C_b : (b_1 \/ b_2 \/ ... \/ ~p \/ ... \/ b_n)

(b_1 \/ ... \/ b_n)

Intuition: modus ponens

p /\ (p -> b_1 \/ ... \/ b_n)

(b_1 \/ ... \/ b_n)

Unit Resolution

C_a : p

C_b : (b_1 \/ b_2 \/ ... \/ ~p \/ ... \/ b_n)

(b_1 \/ ... \/ b_n)

Boolean Constraint Propagation

Unit Resolution

$$\frac{\begin{array}{l} C_a : p \\ C_b : (b_1 \lor b_2 \lor \ldots \lor \sim p \lor \ldots \lor b_n) \end{array}}{(b_1 \lor \ldots \lor b_n)}$$

Boolean Constraint Propagation

```
bcp f =
   case pick_unit_clause f of
   | x => bcp (f[x ↦ ⊤])
   | NONE => f
```

Davis Putnam Logemann Loveland

```
dpll f =
   f' = bcp f
   case f' of
   | ⊤  => SAT
   | ⊥  => UNSAT
   | _  =>
      x = pick_var f'
      if dpll f'[x ↦ ⊤] = SAT then
         SAT
      else
         dpll (f'[x ↦ ⊥])
```

Pure Literal Propagation

If a literal only occurs positively, $\top$

If a literal only occurs negatively, $\bot$

DPLL + PLP

```
dpll f =
   f' = plp (bcp f)
   case f' of
   | ⊤  => SAT
   | ⊥  => UNSAT
   | _  =>
      x = pick_var f'
      if dpll f'[x ↦ ⊤] = SAT then
         SAT
      else
         dpll (f'[x ↦ ⊥])
```